Spatial Geometric Constraint Solving Based on k-connected Graph Decomposition*

Gui-Fang Zhang
School of Sciences
Beijing Forestry University
Beijing 100083, China

Xiao-Shan Gao
KLMM, Institute of Systems Science
Academia Sinica
Beijing 100080, China

ABSTRACT

We propose a geometric constraint solving method based on connectivity analysis in graph theory, which can be used to decompose a well-constrained problem into some smaller ones if possible. We also show how to merge two rigid bodies if they share two or three geometric primitives in a bi-connected or tri-connected graph respectively. Based on this analysis, problems similar to the "double banana problem" could be easily detected.

Categories and Subject Descriptors

J.6 [**COMPUTER-AIDED ENGINEERING**]: CAD

Keywords

Geometric constraint solving, parametric CAD, k-connected graph, separating k-tuple, decomposition

1. INTRODUCTION

Geometric constraint solving(GCS) is one of the key techniques in parametric CAD, which allows the user to make modifications to existing designs by changing parametric values. There are four major approaches to geometric constraint solving: the numerical approach, the symbolic computation approach, the rule-based approach, and the graph-based approach. This paper will focus on using graph algorithms to decompose a large constraint problem into smaller ones.

In [11], Owen proposed a GCS method based on the tri-connected decomposition of graphs. In [1], Hoffmann et al proposed a method based on cluster formation to solve 2D and 3D constraint problems. In [5], Joan-Arinyo et al proposed an algorithm to decompose a 2D constraint problem into an s-tree. This method is equivalent to Owen's and Hoffmann's methods, but is conceptually simpler.

*Partially supported by a 973 project of China

SAC'06 April 23-27, 2006, Dijon, France

The above approaches use triangles as basic patterns to solve geometric constraint problems. In [9], Latham and Middleditch proposed an algorithm which could be used to decompose a constraint problem into what we called *general construction sequence*[2]. A similar method based on maximal matching of bipartite graphs was proposed in [8]. In [4], Hoffmann et al gave an algorithm to find rigid bodies in a constraint problem. From this, several general approaches to GCS are proposed [4]. In [2], a c-tree decomposition method is proposed to solve both 2D and 3D constraint problems. The method proposed in [4] and the c-tree method [2] can be used to find a decomposition with the smallest controlling problem in certain sense. In [13], a method based on connectivity analysis from graph theory is proposed for solving 2D geometric constraint problems. This method is a natural generalization of the methods in [5, 11] which are based on tri-connectivity analysis of the constrained graph, and can solve problems that can be reduced to triangular form.

In this paper, we extend the results in [13] to 3D case. The method works as follows, we first decompose a well-constrained constraint graph into k-connected subgraphs for the smallest possible k. If one of the subgraphs is also well-constrained, then we can solve the original problem by first solving the subgraphs and then merging these subgraphs. The merging process is carefully analyzed. As a result, the "double banana problem" could be easily detected in the algorithm.

In Section 2, we introduce the concept of connected graph. In Section 3, the method to split constraint graph is proposed. In Section 4, an algorithm to generate the D-tree is proposed. In Section 5, a method to merge bi-connected and tri-connected constraint graphs is proposed.

2. PRELIMINARY RESULTS

The geometric primitives considered are points, lines and planes in 3D. The geometric constraints considered include distance constraints between point/point, point/line, point /plane, line/line and the angular constraints between line/line, line/plane, plane/plane in 3D.

We use a *constraint graph* to represent a constraint problem. The vertices of the graph represent the geometric primitives and the edges represent the constraints. Let $G = (V, E, \omega)$(or $G = (V(G), E(G), \omega)$) be a geometric constraint graph. $\forall v \in V$, $\omega(v)$ is the weight of vertex v, i.e the number of independent parameters used to determine the vertex, and $\omega(V) = \Sigma_{v \in V}\omega(v)$. For instance, the weight of every point or plane is 3 in 3D and the weight of a line in 3D is 4. $\forall e \in E$, $\omega(e)$ represents the weight of edge e, i.e

the number of scalar equations to represent the constraint, and $\omega(E) = \Sigma_{e\in E}\omega(e)$. For instance, the weight of the distance constraint between two points is 1 if the distance is not zero, otherwise it is 3. Let $G = (V, E, \omega)$ be a geometric constraint graph.

1. G is structurally over-constrained if there is an induced subgraph $H = (V_H, E_H)$ satisfying $\omega(E_H) > \omega(V_H) - 6$, where $|V_H| > 2$.
2. G is structurally under-constrained if it is not structurally over-constrained and $0 < \omega(E) < \omega(V) - 6$.
3. G is structurally well-constrained if it is neither structurally under-constrained nor over-constrained.

Let $G = (V, E, \omega)$ be a geometric constraint graph in 3D. We define the deficit function associated with G as

$$\text{deficit}(G) = (\omega(V) - 6) - \omega(E).$$

If G is a structurally well-constrained problem, $\text{deficit}(G) = 0$. If G is not structurally over-constrained, $\text{deficit}(G) \geq 0$.

An undirected graph $G = (V, E)$ is called *connected* if for every two nodes $x, y \in V$ there exists a path of edges from E joining x and y. A graph is called disconnected if it is not connected. A graph is called *k-connected* ($k \geq 1$) if there does not exit a set of $k-1$ or fewer nodes $V' \subseteq V$ such that the removal of all nodes of V' and their incident edges from G results in a disconnected graph.

Two vertices x and y of graph G are said to be k-connected if k is the largest integer such that there exist k vertex-disjoint paths from x to y in G. The connectivity of x and y is denoted by $\kappa(x, y)$, which is the maximal number of vertex disjoint paths from x to y in G.

THEOREM 2.1 ([6]). **(Theorem of Whitney)** *A graph G is k-connected if and only if $\kappa(x, y) \geq k$ for any two vertices x and y of G, that is, $\kappa(G) = min\{\kappa(x, y) : x, y \in V\}$.*

The complexity of the algorithm to calculate the connectivity of a connected graph $G = (V, E)$ is $O(|V|^{\frac{1}{2}}|E|^2)$ [6].

THEOREM 2.2. *Let G be a well-constrained graph. We have*

$$\kappa(G) \leq \begin{array}{l} 5 \text{ in 3D for points and planes} \\ 7 \text{ in 3D for points, planes and lines.} \end{array}$$

Proof. For a graph $G = (V, E)$, from [10] it is known that $\kappa(G) \leq \frac{2|E|}{|V|}$. Then for a structurally well-constrained constraint graph $G = (V, E)$, we can obtain the bound of $\kappa(G)$ explicitly as follows.

1. If the primitives are points and planes, then $|E| \leq 3|V| - 6$ and $\frac{2|E|}{|V|} \leq \frac{2(3|V|-6)}{|V|} < 6$. Thus $\kappa(G) \leq 5$.
2. If the geometric primitives are points, lines and planes, then let V_3 be the set of all vertices of weight three, $|E| \leq 3|V_3| + 4(|V| - |V_3|) - 6 = 4|V| - |V_3| - 6$. Thus $\frac{2|E|}{|V|} \leq \frac{2(4|V|-|V_3|-6)}{|V|} < 8$ and $\kappa(G) \leq 7$. ∎

Let $G = (V, E, \omega)$ be a connected undirected graph. A vertex $v \in V$ is a *separating vertex* for G if the induced subgraph by $V - \{v\}$ is not connected. G is bi-connected if it contains no separating vertex.

A pair of vertices $v_1, v_2 \in V$ is a *separating pair* for G if the induced subgraph on $V - \{v_1, v_2\}$ is not connected. G is tri-connected if it contains no separating pairs and vertices[3].

A triplet $\{v_1, v_2, v_3\}$ of distinct vertices in V is a *separating triplet* of a tri-connected graph if the subgraph induced by $V - \{v_1, v_2, v_3\}$ is not connected. G is 4-connected if it contains no separating triplets, pairs and vertices [7].

A tuple $\{v_1, v_2, \cdots, v_k\}$ of distinct vertices in V is a *separating k-tuple* of a k-connected graph if the subgraph induced by $V - \{v_1, v_2, \cdots, v_k\}$ is not connected. G is $(k+1)$-connected if it contains no separating n-tuple $\{v_1, v_2, \cdots, v_n\}$ $(n = 1, \cdots, k)$.

3. SPLIT AND CUT GRAPHS

In this section, let $G = (V, E, \omega)$ be a structurally well-constrained geometric constraint graph. A subgraph G_s in G is called a *cut graph* of G if G is not connected after deleting G_s and the edges adjacent to vertices in G_s. The ways to find cut graphs of G are as follows.

1. Assuming that G is k-connected and $\{v_1, v_2, \cdots, v_k\}$ is a separating k-tuple, where $v_i \in V (i = 1, \cdots, k)$. The subgraph of G induced by $\{v_1, v_2, \cdots, v_k\}$ is a cut graph.
2. Assuming that H is a structurally well-constrained subgraph in G, i.e. $\text{deficit}(H) = 0$, let

   $$V_c = \{v | (v, v_1) \in E(G), v \in V(H), v_1 \in (V(G) - V(H))\},$$

   if $|V(H)| > |V_c|$, the subgraph of G induced by V_c is a cut graph associated with H.

A k-connected graph can be split into *split components* by splitting it at the separating k-tuple, i.e. deleting the separating k-tuple and edges adjacent to the k-tuple.

Let $V_s = \{v_1, v_2, \cdots, v_k\}$ be the separating k-tuple in the graph G that induces the split components $C^1, C^2, \cdots, C^n$.

If there exists a split component $C^j (0 \leq j \leq n)$ satisfying $\text{deficit}(C^j) = 0$ and we can find a cut graph G_s associated with C^j. Let $G_1 = C^j$ and G_2 the subgraph of G induced by $(V(G) - V(C^j)) \bigcup V_s$. G_1 and G_2 are called the *split graphs* of G.

For each $C^i (i = 1, \cdots, n)$, if we duplicate V_s to each C^i, the subgraph of G induced by $V_s \bigcup V(C^i)$ is called *split component* and denoted by C_i.

If there exists a split component $C_k (0 \leq k \leq n)$ satisfies $\text{deficit}(C_k) = 0$, we can find a cut graph G_s associated with C_k. Let $G_1 = C_k$ and $G_2 = \bigcup_{i=1, i\neq k}^{n} C_i$. G_1 and G_2 are called the *split graphs* of G.

For each $C_i (i = 1, \cdots, n)$, if we can not find a cut graph associated with C_i, select an m such that $1 \leq m \leq n$ and let

$$G_1 = \bigcup_{i=1}^{m} C_i; \quad G_2 = \bigcup_{i=m+1}^{n} C_i. \tag{1}$$

The graph G_1 and G_2 are called the *split graphs* of G.

The relation of split graphs and the cut graph is shown in Fig.1.

THEOREM 3.1. *Let G be a k-connected structurally well-constrained graph, $G_s = (V(G_s), E(G_s), \omega)$ the cut graph, $G_1 = (V(G_1), E(G_1), \omega)$ and $G_2 = (V(G_2), E(G_2), \omega)$ the split graphs. We have*

$$\textit{deficit}(G_1) + \textit{deficit}(G_2) = \textit{deficit}(G_s). \tag{2}$$

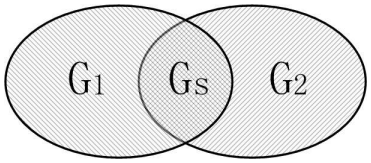

Figure 1: Relation between split graphs and cut graph

Proof: Since G is structurally well-constrained, deficit$(G_s) \geq 0$, deficit$(G_1) \geq 0$ and deficit$(G_2) \geq 0$. If deficit$(G_s) > 0$, deficit$(G_1) > 0$ and deficit$(G_2) > 0$, then $\omega(V_s) - E_s - D > 0$, $\omega(V(G_1)) - \omega(E(G_1)) - D > 0$ and $\omega(V(G_2)) - \omega(E(G_2)) - D > 0$.
Because
$\omega(E(G_1)) + \omega(E(G_2)) - \omega(E_s) = \omega(V) - D$,
we have $\omega(E(G_1)) + \omega(E(G_2)) - \omega(E_s)$
$= \omega(V(G_1)) + \omega(V(G_2)) - \omega(V_s) - D$.
Then $(\omega(V(G_1)) - \omega(E(G_1)) - D) + (\omega(V(G_2))$
$- \omega(E(G_2)) - D) = \omega(V_s) - \omega(E_s) - D$.
Thus deficit(G_1) + deficit(G_2) = deficit(G_s). ∎

COROLLARY 3.2. *Let $G = (V, E, \omega)$ be a structurally well-constrained k-connected graph, $G_s = (V_s, E_s, \omega)$ the subgraph of G induced by a separating k-tuple $V_s = \{v_1, v_2, \cdots, v_k\}$, G_1 and G_2 the split graphs of graph G. G_1 and G_2 are structurally well-constrained if and only if G_s is structurally well-constrained.*

COROLLARY 3.3. *Let G_s be the graph induced by a separating k-tuple of a k-connected structurally well-constrained graph G, G_1 and G_2 the split graphs. If G_s is not structurally well-constrained and G_1 is structurally well-constrained, then deficit(G_2) = deficit(G_s).*

In general, a structurally well-constrained graph can be decomposed by the following ways based on connectivity analysis.

1. deficit$(G_s) = 0$. Now the split graphs G_1 and G_2 are structurally well-constrained by Corollary 3.2. We can solve them separately, then merge them to obtain the solutions to the initial problem. Fig. 2 is an example of this case. The graph in 3D is split into two structurally well-constrained graphs.

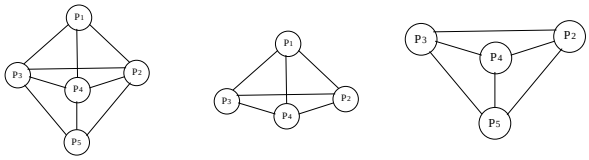

Figure 2: Separating triplet is P_2, P_3, P_4 and deficit$(G_s) = 0$

2. deficit$(G_s) > 0$. According to theorem 3.1, deficit(G_s) = deficit(G_1) + deficit(G_2) and deficit(G_1) can be 0, 1, $\cdots$, deficit(G_s). If either G_1 or G_2 is structurally well-constrained, then let G_1 be the well-constrained split graph. We solve G_1 first, then add deficit(G_s) *auxiliary constraints* to G_s so that G_s becomes a structurally well-constrained problem G'_s. Now, G_2 also becomes a well-constrained problem G'_2, and can be solved separately. After getting the solution of G_1 and G'_2, we can merge them to obtain a solution to the initial problem. G'_2 is called the *modified split graph* of G with G_1.

   If neither G_1 nor G_2 is structurally well-constrained, we can make the following choices:

   (a) Select another separating k-tuple and re-decompose the constraint graph G; or
   (b) Solve G with numerical computation.

In the cases deficit$(G_s) > 0$, we need to add auxiliary edges to make the cut graph G_s well-constrained. This a *well-constrained completion* problem. Latham et al presented a method to solve such a problem [9].

4. A DECOMPOSITION ALGORITHM

We will introduce a new decomposition tree, D-tree, which can be used to simplify a structurally well-constrained constraint problem.

A *D-tree* for a structurally well-constrained graph $G = (V, E)$ is a binary tree.

1. The root of the tree is the graph G. Left child L and right child R are defined as follows.
2. For each node N in the tree, its left child L is the split graph of N which is either a triangle or a structurally well-constrained subgraph of N, and the right child R is the (modified) split graph of N with L which is either a triangle or a structurally well-constrained graph.
3. All leaves are either a triangle or a structurally well-constrained j-connected ($3 \leq j$) constraint graph.

ALGORITHM 4.1. *The input is a structurally well-constrained graph $G = (V(G), E(G), \omega)$ in 3D. The output is a D-tree for G. Let T=G as the initial value, and $S_k = \emptyset$ as the initial value of set of separating k-tuples in $V(T)$.*

S1 *If T is a triangle, the algorithm terminates; else goto step* **S2**.

S2 *Calculate connectivity k of the connected graph T. If $|V(T)| - k < 2$, the algorithm terminates; else let $S_k \leftarrow \emptyset$ goto* **S3**.

S3 1. *If $k = 2$, find all the separating pairs with Hopcroft and Tarjon's method [3]. Then add these separating pairs to S_k, goto step* **S4**.
   2. *If $k = 3$, find all the separating triplets with the method in [7]. Then add these separating triplets to S_k, goto step* **S4**.
   3. *If $k > 3$, goto step* **S5**.

S4 *If $S_k \neq \varnothing$, taking a separating pair or triplet $S \in S_k$, $S_k \leftarrow S_k - \{S\}$, generate the cut graph G_s induced by S, goto* **S7**. *Otherwise, the algorithm terminates.*

S5 *Let $H = T$. For all k-tuple $S = \{v_1, \cdots, v_k\}$ in H, do step S6. If we cannot find a separating t-tuple, the algorithm terminates.*

S6 *Let U be the graph induced by $H - S$. If U is not connected, S is a separating k-tuple. Generate the cut graph G_s induced by S, goto* **S7**. *If U is connected, goto step S5.*

S7 *Splitting T by the cut graph to generate the split graphs G_1 and G_2. If the deficit function of the cut graph induced by S is 0, goto* **S8**; *else goto* **S9**.

S8 *Let $L = G_1$ and $R = G_2$. Let T=L, repeat the algorithm from* **S1** *recursively. Let T=R, repeat the algorithm from* **S1** *recursively.*

S9 *If one of the split graph is structurally well-constrained, let it be G_1 and goto* **S10**; *else if neither G_1 nor G_2 is structurally well-constrained, goto* **S11**.

S10 *Let $L = T = G_1$ and operate the algorithm from* **S1** *recursively for T. Add deficit(G_s) auxiliary constraints to G_2 to make the modified split graph G_2^m structurally well-constrained with the method proposed in [9]. Let R be the modified split graph G_2^m, and $T = R$. Repeat the algorithm from* **S1** *recursively.*

S11 *Check the separating component $C^1, C^2, \cdots, C^n$. If we can find a cut graph G_s associated with $C^j (0 \leq j \leq n)$, goto* **S12**; *else if $k \leq 3$ goto* **S4**; *else if $k > 3$ goto* **S5** *to repeat the process for another k-tuple.*

S12 *Generate the split graphs G_1 and G_2. If deficit(G_s) = 0, goto* **S8**; *else goto* **S10**.

Let $n = |V|$. The complexity of step **S2** is $n^{\frac{1}{2}}|E|^2$[6]. When $k = 2$, the complexity of **S3** is $O(n + |E|)$ [3]; when $k = 3$, the complexity of **S3** is $O(n^2)$ [7]. For Step **S5**, in the worst case, we need to consider $\frac{|n!}{k!(n-k)!}$ k-tuples. From Theorem 2.2, we know $k \leq 7$. Then the complexity is $O(n^7)$ at most to get all the separating k-tuples. So, the complexity to find the first split is $O(n^7)$. We need to repeat the algorithm for the two split graphs with v_1 and v_2 vertices ($v_1 + v_2 = n$). Then the split process for the two split graphs are $O(v_1^7 + v_2^7) \leq O(n^7)$. So the complexity of the algorithm is at most $O(n^8)$, and is hence polynomial in n.

5. MERGE SPLIT GRAPHS

After a D-tree is obtained for a geometric constraint problem, we can solve the problem as follows: Do a left to right depth first transversal of the D-tree and solve the constraint problem represented by each node as follows.

1. If the current node N is a leaf in the tree then it is a well-constrained problem that cannot be decomposed further with the algorithm. Solve N with numerical computation methods.
2. Let N be a node with left child L and right child R. This can be done in three steps.
   (a) **Solve the left child** L. L is a well-constrained problem which can be solved recursively.
   (b) **Solve the right child** R. The values for the parameters in the auxiliary constraints in the right child R can be obtained from L. Now, R is a well-constrained problem which can be solved recursively.
   (c) **Merge L and R to obtain N.**

In what below, we will give a detailed analysis of the merging process for bi-connected and tri-connected graphs. The general case can be reduced to one of these cases. The reason is that the relative position of two rigid bodies can be fixed by merging them along a minimal rigid body shared by them, and a minimal rigid body contains two or three primitives [2].

5.1 Merge bi-connected constraint graphs

Let the separating pair be $\{a, b\}$ in a bi-connected graph G. The split subgraphs of G are two rigid bodies R_1 and R_2. Now we show how to assemble R_1 and R_2.

The problem can be classified into the following six cases according to the types of a and b.

1. The vertices a and b are two points. The relative position of R_1 and R_2 can not be fixed because R_1 may rotate around the segment ab assuming that R_2 is fixed. Thus $R_1 \bigcup R_2$ is not a rigid body anymore although structurally well-constrained
2. The vertices a and b are two planes. There is at least one translation degree of freedom between R_1 and R_2. Thus $R_1 \bigcup R_2$ is not a rigid body.
3. The vertices a and b are a point and a plane. The relative position of R_1 and R_2 can not be fixed because there is a rotation degree of freedom left. Thus $R_1 \bigcup R_2$ is not a rigid body anymore.
4. The vertices a and b are two lines. If a and b are parallel, the relative position of R_1 and R_2 can not be fixed. If a and b are not parallel, the relative position of R_1 and R_2 can be fixed.
5. The vertices a and b are a point and a line. If the distance between a and b is zero, the relative position of R_1 and R_2 can not be fixed. If the distance between a and b is not zero, the relative position of R_1 and R_2 can be fixed.
6. The vertices a and b are a plane and a line. When a and b are parallel, the relative position of R_1 and R_2 can not be fixed. If a and b are perpendicular to each other, the relative position of R_1 and R_2 can not be fixed. Let the angle between a and b be $\angle(a, b)$, if $0 < |\cos(\angle(a, b))| < 1$, the relative position of R_1 and R_2 can be fixed.

THEOREM 5.1. *Let $\{a, b\}$ be the separating pair in a bi-connected graph G in 3D, R_1 and R_2 the split subgraphs of G which are two rigid bodies. Then R is a rigid body if and only if $\{a, b\}$ are of the following three cases: a point and a line which are not incident; a plane and a line which are not parallel, perpendicular or incident to each other; two lines which are not parallel to each other.*

The problem in Fig. 3(a) is the double banana problem, where the weight of each vertex is 3 and the weight of each edge is 1. $\{p_1, p_5\}$ is a separating pair. G_1 induced by $\{p_1, p_2, p_3, p_4, p_5\}$ and G_2 induced by $\{p_1, p_5, p_6, p_7, p_8\}$ are structurally well-constrained. But the cut graph induced by $\{p_1, p_5\}$ is not structurally well-constrained. According to Theorem 5.1, G is not a rigid body if G_1 and G_2 are rigid bodies. For a recent work to characterize such ill-conditioned problems, please consult [12].

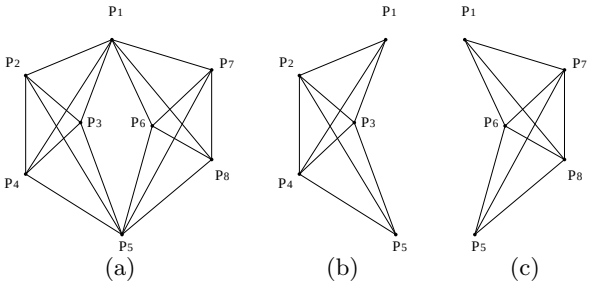

Figure 3: Double banana problem

5.2 Merge tri-connected constraint graphs

Let $V_s = \{a, b, c\}$ be the separating triplet of $G = (V, E, \omega)$, $G_s = (V_s, E_s, \omega)$ the subgraph induced by V_s, G_1 and G_2 the split graphs which are rigid bodies. We will try to assemble R_1 and R_2. The problem can be classified into the following ten cases according to the types of a, b and c.

1. The vertices a, b and c are three non-collinear points. The relative position of R_1 and R_2 can be fixed and $R_1 \bigcup R_2$ is a rigid body.
2. a, b and c are three planes. If two of them are parallel, the relative position of R_1 and R_2 can not be fixed. If the three planes intersect with each other and the three intersection lines are parallel, the relative position of R_1 and R_2 can not be fixed. Otherwise the relative position of R_1 and R_2 can be fixed.
3. The vertices a, b and c are three lines. If the three lines are parallel to each other, the relative position of R_1 and R_2 can not be fixed. Otherwise the relative position of R_1 and R_2 can be fixed.
4. The vertices a and b are two points and c is a plane. The relative position of R_1 and R_2 can be fixed.
5. The vertices a and b are two points and c is a line. If both the two points are on the line, the relative position of R_1 and R_2 can not be fixed. Otherwise the relative position of R_1 and R_2 can be fixed.
6. The vertex a is a point, b and c are two planes. The relative position of R_1 and R_2 can be fixed.
7. The vertex a is a point, b and c are two lines. The relative position of R_1 and R_2 can be fixed.
8. a and b are two planes and c is a line. If $a \parallel b$ and the line is perpendicular to the planes, the relative position of R_1 and R_2 can not be fixed. If the line is parallel to the two planes, the relative position of R_1 and R_2 can not be fixed. Otherwise the relative position of R_1 and R_2 can be fixed.
9. The vertices a and b are two lines and c is a plane. If the two lines are parallel and on the plane, the relative position of R_1 and R_2 can not be fixed. Otherwise the relative position of R_1 and R_2 can be fixed.
10. Vertex a is a point, b is a plane, and c is a line. If a is on c and c is perpendicular to b, the relative position of R_1 and R_2 can not be fixed. Otherwise, the relative position of R_1 and R_2 can be fixed.

For k-connected graph ($k > 3$), we can always select three geometric primitives in the separating k-tuple to merge the two rigid bodies with the three primitives.

6. REFERENCES

[1] I. Fudos and C.M. Hoffmann, A Graph-constructive Approach to Solving Systems of Geometric Constraints, *ACM Transactions on Graphics*, **16**(2), 179-216, 1997.

[2] X.S. Gao and G.F. Zhang, Geometric Constraint Solving via C-tree Decomposition, *Proc.ACM SM03*, 45-55, ACM Press, New York, 2003.

[3] J.E. Hopcroft and R.E. Tarjan, Dividing a Graph into Triconnected Components, SIAM J. Comput. 135-158, 1973.

[4] C.M. Hoffmann, A. Lomonosov and M. Sitharam, Decomposition Plans for Geometric Constraint Systems, II: New Algorithms, **31**, 409-427, *J. of Symbolic Computation*, 2001.

[5] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta and J. Vilaplan-Pastó, Revisiting Decomposition Analysis of Geometric Constraint Graphs, *Proc. ACM SM02*, 105-115, ACM Press, New York, 2002.

[6] D. Jungnickel, *Graphs, Graphen, Netzwerke, und Algorithmen*, Springer, Berlin, 1999.

[7] A. Kanevsky and V. Ramachandran, Improved Algorothms for Graph Four-connectivity, *Proc. 28th Ann. IEEE Symp. Foundations of Computer Science*, Los Angeles, 252-259, 1987.

[8] H. Lamure and D. Michelucci, Qualitative Study of Geometric Constraints, in *Geometric Constraint Solving and Applications*, 234-258, Springer, Berlin, 1998.

[9] R.S. Latham and A.E. Middleditch, Connectivity Analysis: a Tool for Processing Geometric Constraints, *Computer-Aided Design*, **28**(11), 917-928, 1994.

[10] J. Van Leeuwen, Handbook of Theoretical Computer Science(Volume A): Algorithms and Complexity, Elsevier Science Publishers B.V. 1990.

[11] J. Owen, Algebraic Solution for Geometry from Dimensional Constraints, in *ACM Symp., Found of Solid Modeling*, ACM Press, New York, 397-407, 1991.

[12] M. Sitharam and Y. Zhou, A Tractable, Approximate Characterization of Combinatorial Rigidity in 3D, Automated Deduction in Geometry, ADG 2004, September, 2004Florida University.

[13] G.F. Zhang and X.S. Gao, Geometric Constraint Solving Based on Connectivity of Graph, *Computer-Aided Design and Applications*, **1**(1-4), 469-476.